N94- 35631

# Overview of the MSTI 2 Processor-In-The-Loop Simulator

Christopher A. Rygaard

Integrated Systems, Inc.

## Abstract

To thoroughly test the on-board software for the MSTI 2 spacecraft, it was necessary to generate an environment for the software which accurately simulated the on-orbit conditions of the spacecraft. To achieve this, the MSTI 2 Processor-In-the-Loop (PIL) high-fidelity simulator was developed. The entire development was completed in 3 months, and required 4 man-months of effort. This paper describes the design and development of this simulator, and the methodology employed.

## Introduction

Thorough testing of the MSTI 2 on-board software required that the software be placed in an environment which accurately reproduces the conditions which the software will encounter while it is on orbit. This was achieved by using a flight processor with flight I/O boards, in conjunction with an AC-100 real-time simulator (see Figure 1). The unmodified on-board software was loaded onto the flight processor, and the I/O boards were utilized in their flight configuration. The AC-100 captured the output signals from the I/O boards, updated its simulation accordingly, and emitted the input signals to the flight processor. This process occurred in real time.

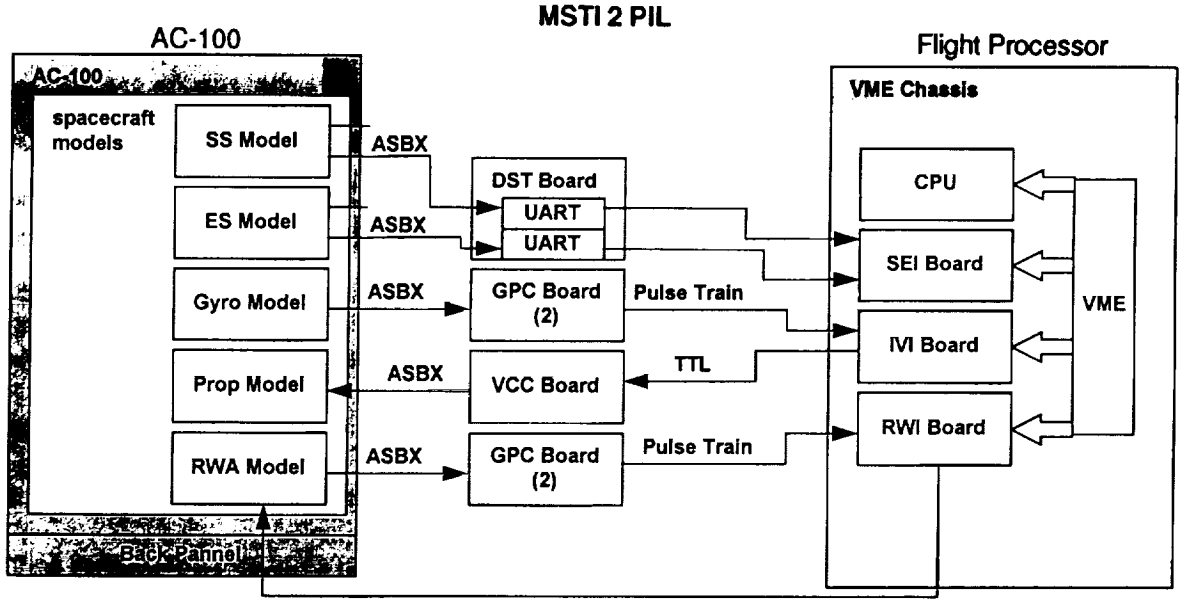


**Figure 1**

This paper discusses the design of the PIL, including an architectural overview, and the development of the PIL, including the methodology which was employed for rapid development. This paper focuses on the

real-time simulator, running on the AC-100, which emulated the on-orbit environment. The flight processor and the on-board software are not discussed in detail in this paper.

## Overview

The MSTI 2 PIL provided a real-time simulation of the spacecraft environment for testing the on-board software running real-time on the flight processor (see Figure 2). The primary task of the MSTI 2 on-board software was attitude control, so the PIL was limited primarily to those functions relating to the Attitude Control System (ACS). The subsystems which were emulated by the AC-100 include attitude dynamics, ACS sensors and actuators, and orbit dynamics.

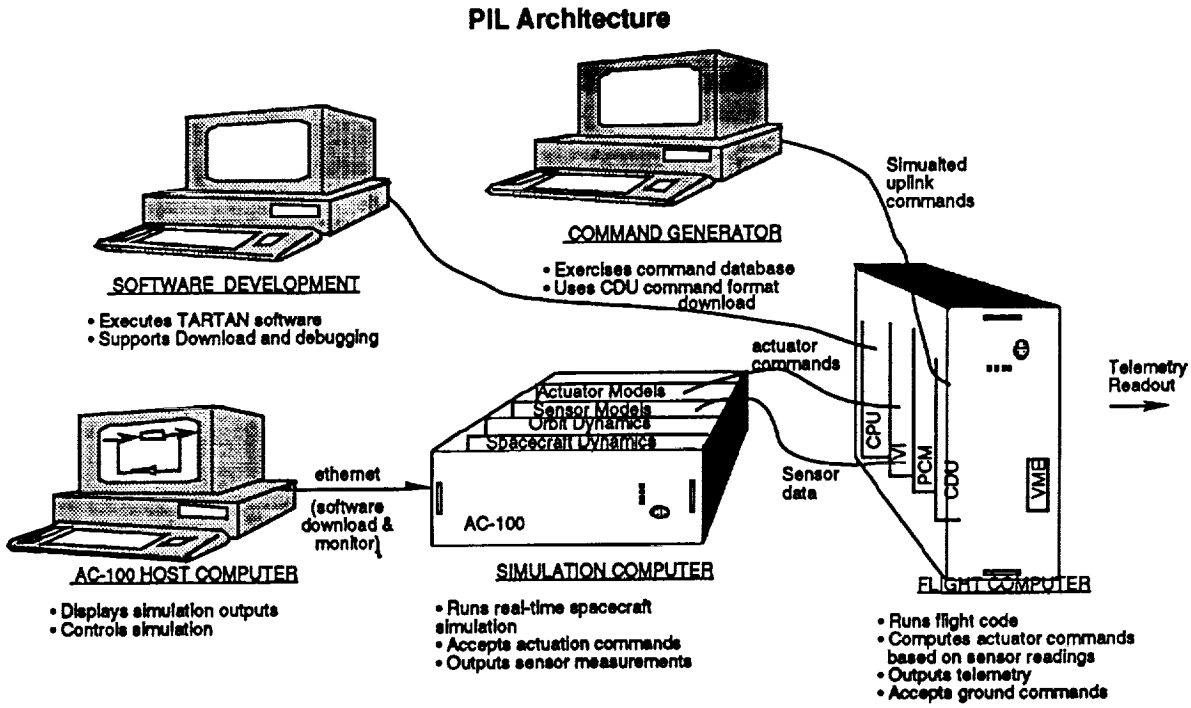


**Figure 2**

The PIL was designed to provide a realistic environment for the on-board software which accurately emulates the interactions between the processor and the rest of the spacecraft. The interfaces between the processor and the AC-100 were restricted to the ACS sensors and actuators, because these are the only ACS interfaces on the spacecraft available to the flight processor. The AC-100 intercepted those commands generated by the flight processor which were intended for the ACS actuators, and passed these commands to the spacecraft models. These models processed the commands, and the propagated dynamical subsystems, in order to generate realistic ACS sensor data. This data was passed along to the AC-100 output hardware, which emulated the electrical characteristics of the sensors.

The simulator hardware includes the AC-100 off-the-shelf real-time simulator, the custom I/O boards for the AC-100, and the host workstation. The simulator software includes the development environment, the automatically generated software, and the handwritten C code. In addition, a dynamics analysis software program, AutoLev, was used to develop the attitude equations of motion and generate the attitude dynamics subroutines.

## The AC-100 System

As indicated in Figure 2, the hardware in the PIL consists of the AC-100 real-time simulator, the host workstation, and the custom I/O electronics boards in the AC-100. The PIL software consists of the Matrix$_x$/SystemBuild development environment, the automatically generated C code, the custom hardware

interface routines, and the C code reused from other projects.

Matrix$_x$/SystemBuild is a single program which provides two environments: Matrix$_x$ and SystemBuild. Matrix$_x$ provides a command line environment for numerical design and analysis, while SystemBuild is an environment which allows the user to model systems with block diagrams, and then simulate the systems directly from the block diagrams. The results of the simulations can be analyzed in the Matrix$_x$ environment.

As an example, Figure 3 shows a block diagram from the earth sensor subsystem. The data flow is indicated by the interconnections between the blocks, and the operations on the data are indicated by the blocks. Figure 3 shows gain blocks, data path switches, logic blocks, trigonometric blocks, and others. Many other blocks are available in the SystemBuild environment. The block labeled "ES Blanking Logic" is a Superblock, and within it is an entire subsystem, which is in turn built of blocks and superblocks. The superblocks can be nested in this way without limit.

**Earth Sensor Block Diagram**

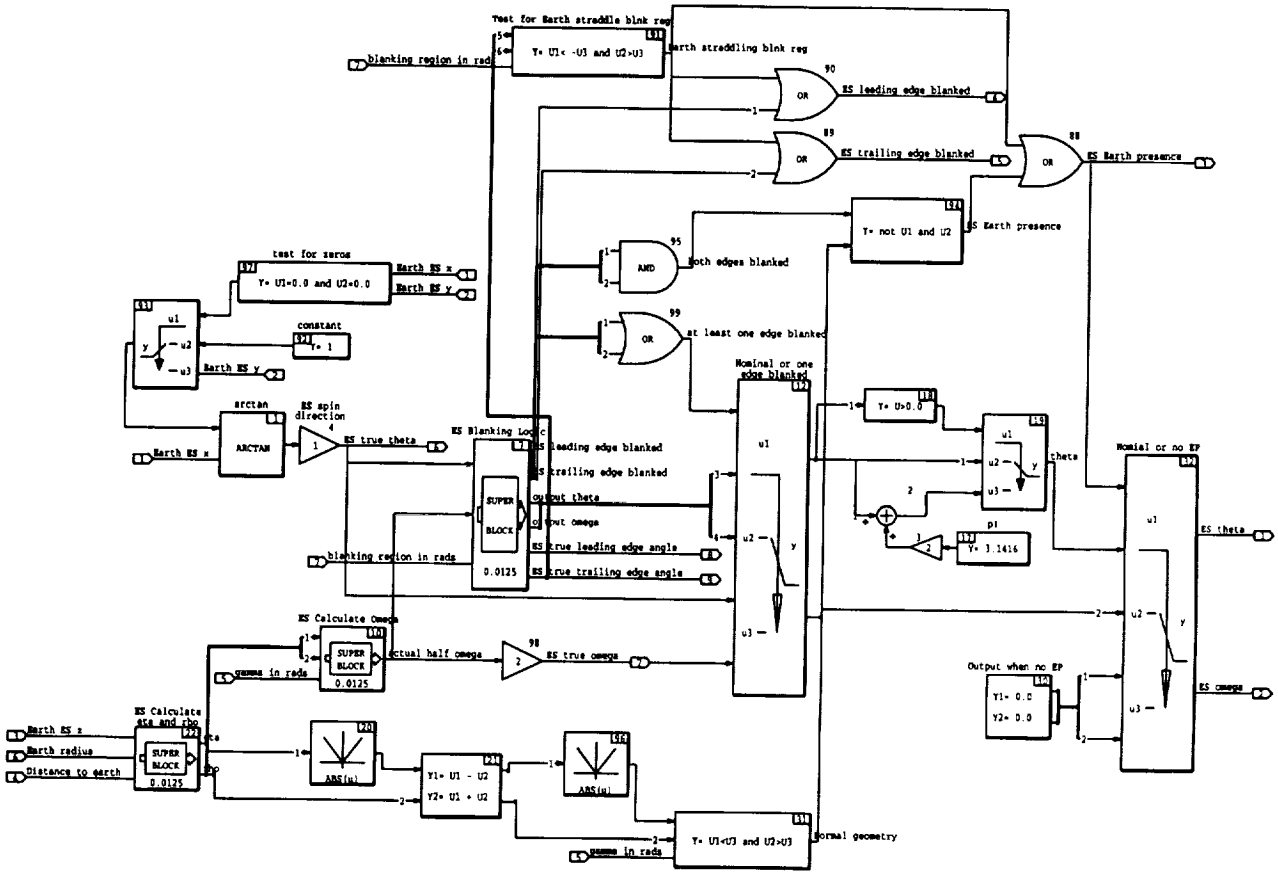


**Figure 3**

AutoCode converts the block diagrams developed under SystemBuild directly into real-time executable C code, which can be downloaded to the AC-100 real-time processor, or any other real-time processor. IA is a graphical environment which allows the user to build control panels to provide real-time interaction with the simulations executing on the AC-100.

The development of the PIL simulator software began under the Matrix$_x$/SystemBuild environment, starting with the MSTI 1 PIL models. In this environment, the various S/C subsystems were modeled using block diagrams. While in this environment, individual subsystems, or the entire spacecraft model, could be simulated non-real-time. Because the block diagrams could be simulated on the host workstation, it was possible to develop and debug the spacecraft models without needing to develop source code, and without porting the software to the AC-100 platform.

Not all of the spacecraft models were in block diagram form. The orbit dynamics models and the sun ephemeris had already been developed on earlier projects in handwritten C code, so it was not desirable to redevelop them. The SystemBuild a capability, called "User Code Blocks", which provides a standardized interface so that the user's handwritten C code can be used within the block diagrams. This provides a migration path from handwritten software to automatically generated simulations. The existing orbit dynamics models and sun ephemeris were incorporated into the block diagram this way.

Once the spacecraft block diagrams were developed and debugged, they were ready for the code generation process for execution on the AC-100. AutoCode converted the block diagrams directly into executable real-time source code, so moving these block diagrams to the AC-100 required no more than a few minutes.

Certain portions of the I/O were also readily ported to the AC-100, because the standard I/O capabilities of the AC-100 are tightly integrated into the software, so that connecting the AC-100 standard I/O was no more difficult than editing the block diagrams.

Other portions of the MSTI 2 PIL I/O had to emulate the specialized interfaces of the various ACS sensors and actuators, which required custom hardware interfaces to be developed for the AC-100. These hardware interfaces required the development of routines to allow the automatically generated code to communicate with the custom-designed I/O boards. These were 5 short C subroutines which called low-level I/O routines which are supplied with the AC-100. These custom routines were developed using the ordinary compile-link-debug cycle.

In addition, the equations of motion of the spacecraft were developed using AutoLev. This software package develops the equations of motion of a dynamical system using Kane's method, and automatically generates source code to simulate these equations. This code was integrated into the SystemBuild models as a User Code Block.

## Spacecraft Subsystem Models

The models which were implemented in the MSTI 2 PIL included attitude dynamics, a sun sensor, an earth sensor, low rate gyros, a high rate gyro, thrusters, reaction wheels, orbit dynamics, and sun ephemeris.

The PIL setup operated at 80 Hz, which provided a sufficiently fast response to the on-board code which was operating at 5 Hz. Throughput testing started by running the models at 20 Hz, and then increasing the rate. The rate was increased to 120 Hz, and the AC-100 had not overflowed, so testing was stopped. It was decided that operating the model at 80 Hz. would allow a growth in model complexity of 50% or more, and this rate was more than sufficient for a quick response to the 5 Hz. on-board software.

The 80 Hz. rate of the PIL was also chosen because the on-board software operated at 5 Hz., it was not necessary to model any phenomena much faster than this. No phenomena which responded faster than about 0.05 seconds was modeled, and the 80 Hz. rate supported this.

## Sun Sensor

The MSTI 2 sun sensor had a square field of view, 64° by 64° in extent. The output of this sensor was two numbers representing the angular position of the sun in the Field Of View (FOV), a single bit to indicate the sun is present in the FOV, and some housekeeping information. This information was encoded in a data frame which was transmitted 5 times per second to the processor. This was transmitted on an ordinary serial data stream, using RS-422 voltage levels. No commands were transmitted from the processor to the sun sensor.

Figure 4 shows the highest level block diagram of the Sun Sensor model. This model used the output of the attitude dynamics models, along with the sun ephemeris model, to compute the sensor data. The model had to check for sun presence in the SS FOV. If the sun was in the field of view, the angles were calculated, otherwise the default values were used. In addition, the "sun present" bit had to be set properly. The housekeeping information was hard-coded to its default value, and was not variable.

The error models included in the sun sensor model included geometric misalignment of the sensor on the spacecraft, and a bias on each output angle.

The data bytes for the serial data stream were formatted in the SystemBuild models, and passed on to the serial output of the AC-100.

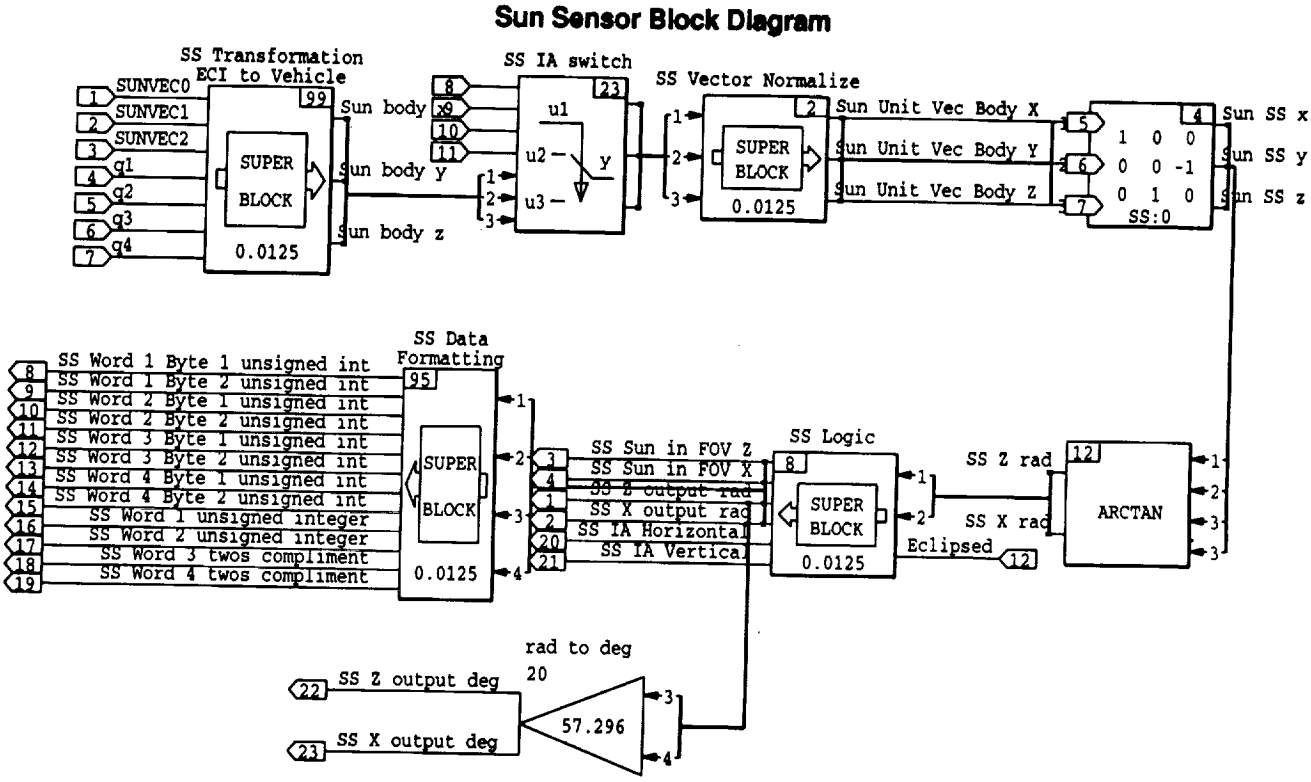


**Figure 4**

## Earth Sensor

The MSTI 2 earth sensor was a scanning horizon sensor with a 60° half-cone angle. Due to the mechanical structure of the sensor, 81° of the scan cone were obstructed.

The outputs of this sensor included two angles, representing the phase and the chord of the earth in the scan cone. In addition, there were three individual informational bits: one bit indicated the earth was present in the scan cone, another single bit indicated that the leading edge of the earth chord was blocked by the obstruction, and the third single bit indicated that the trailing edge of the earth chord was blocked by the obstruction. This information, along with some housekeeping information, was encoded in a data frame which was transmitted 5 times per second to the processor. This was transmitted on an ordinary serial data stream using RS-422 voltage levels. No commands were transmitted from the processor to the earth sensor.

Figure 3 shows the block diagram of the Earth Sensor model. This model used the output of the attitude dynamics model, along with the orbit dynamics model, to compute the sensor data. If the scan cone intersected the earth, the chord and phase were computed, and the three bits described above, were set appropriately. If the scan cone did not intersect the earth, the default values for the angles were used, and the default values were used for the blanking bits. The housekeeping information was hard-coded to its default value, and was not variable.

The error models for the earth sensor included geometric misalignment of the sensor on the spacecraft, and a bias on each of the angle outputs.

The data bytes for the serial data stream were formatted in the SystemBuild models, and passed on to the serial output of the AC-100.

## Low Rate Gyro

The primary source of attitude rate information on the MSTI 2 spacecraft was two high accuracy 2-axis gyros. However, they would saturate at a relatively low angular rate.

Each sensitive axis of each gyro had two pulse streams: one for positive rotation, and one for negative rotation. Only one of these two pulse streams was active at any moment. The frequency of these pulse streams was proportional to the angular rate about the sensitive axis. In addition, each pulse stream had a redundant transmitter for reliability. These pulse streams were transmitted on a RS-422 differential pair. There were a total of 32 conductors transmitting the gyro data.

There was no housekeeping information transmitted from the gyros to the processor, and there were no commands transmitted from the processor to the gyros.

The gyro models on the AC-100 used data only from the attitude dynamics models. The error models included geometric misalignment of the gyro on the spacecraft, and gyro biases.

Because the gyros had to be emulated with very high precision, and because the gyro outputs were specialized, it was necessary to build a custom output board for the AC-100, the GPC board. This board provided closed-loop control of the frequency. A counter was placed on each pulse stream, and these counters could be read by the AC-100 to provide a feedback path. The angular rate of the spacecraft about each gyro axis was integrated over time, and this accumulated value was compared to the counters during each 80 Hz. step. The frequency of each step was adjusted to make sure that the gyro outputs were never more than one pulse in error, when compared against the software models. The feedback path allowed the GPC board to generate smooth pulse trains which were lower than 80 Hz., by shutting down the pulse stream altogether during one or more 80 Hz. periods, and only occasionally commanding one pulse to be transmitted.

Each GPC board could transmit all of the signals coming from one gyro, so 2 GPC boards were required in the PIL to emulate these gyros.

It was necessary to develop a small handwritten C routine to drive the GPC boards. This routine handled the feedback control of the GPC board. This routine took the integrated angular rate from the block diagrams, and read the feedback counters on the GPC boards. After applying appropriate scale factors, the two values were compared for errors. Any differences were compensated by adjusting the frequency during the next 80 Hz cycle. With this feedback scheme, the accumulated errors of the gyro emulators, compared to the desired values, never exceeded 8 arcseconds.

This interface routine interacted with the hardware by calling simple low-level I/O routines, which are bundled with the AC-100. This routine was incorporated into the SystemBuild block diagram as a User Code Block.

## High Rate Gyro

The MSTI 2 spacecraft included one low fidelity 3-axis gyro which had favorable saturation characteristics in order to handle that portion of the mission in which the spacecraft had high angular rates. The output of this gyro was three voltage levels, one for each axis. The voltage level was proportional to the angular rate of the spacecraft about the respective axis. There was no housekeeping information transmitted by this gyro, and there were no commands from the flight processor to this gyro.

The high rate gyro model used data from the attitude dynamics model to compute the gyro output. The standard AC-100 configuration includes several digital-to-analog outputs, so emulating the gyro electrical interface was quite simple. Using the standard I/O connection editor, this entire model, along with its I/O, required about one half of a day to implement.

## Reaction Wheel Assembly

The MSTI 2 spacecraft had three reaction wheels, one along each primary axis of the spacecraft. These reaction wheels accepted an analog electrical signal as a torque command from the flight processor, and emitted an electrical pulse stream whose frequency was proportional to the wheel speed. In addition, there were various housekeeping commands which the processor could transmit to the wheels, and there were

various housekeeping signals which the wheels could transmit to the processor.

The reaction wheel models incorporated two torque sources. First, the motor torque was directly proportional to the analog torque command, so this was modeled by a simple gain. Second, the frictional torque varied as a function of wheel speed, and this was modeled using a simple table lookup block. The net torque acting on the wheel was the signed sum of these two torques.

The net torque computed by the wheel model was passed along to the attitude dynamics model, which would compute the wheel angular accelerations relative to the spacecraft. The reaction wheel model would integrate this acceleration to determine the wheel speed.

The standard AC-100 configuration includes several analog-to-digital inputs, so the input to this model was quite easy to implement. The output from this model was a pulse stream. While it was not necessary to control this pulse stream with high accuracy, the pulse stream generator from the gyro subsystem had already been developed, so it was easiest to simply reuse the gyro software and hardware with only changes in a few parameters in the software. Each GPC board could emulate two reaction wheels, so two additional copies of this board were required to emulate the RWAs. The housekeeping inputs and outputs of the reaction wheels were not modeled in the PIL.

## Thrusters

The MSTI 2 spacecraft included 12 thrusters: 8 low-force thrusters for attitude control and 4 high-force thrusters for orbit adjust and orbit maintenance. These were simple on-off thrusters, and could not be throttled for proportional control. The flight processor issued no commands directly to the thrusters. Instead, the propulsion valves were controlled by the processor. A high TTL level signal opened the valves, and a low TTL level signal closed the valves. There were many housekeeping signals to and from the spacecraft propulsion system.

The fuel system on the MSTI 2 spacecraft regulated the pressure of the fuel being fed to the thrusters.

The interface board in the flight processor could command a thruster burn duration in increments of 250 μs. These TTL level signals were issued by the on-board software once per 5 Hz. period. The MSTI 2 thrusters had a very short thrust buildup at the beginning of each thrust pulse, followed by a very short thrust tail-off at the end of each thrust pulse.

The thruster models in the MSTI 2 PIL were modeled with no thrust buildup or tail-off. The models simply applied a force and a moment on the spacecraft, based on the thrust capability and location of the individual thrusters. The forces were summed and passed along to the orbit dynamics models, and the torques were summed and passed along to the attitude dynamics models. The propulsion models did not include blowdown of the fuel system, because this was a pressure regulated system.

In order to maintain sufficient fidelity of the attitude dynamics, it was decided that the thruster commands should be captured with a resolution finer than one 80 Hz. period. This required a custom interface board, the Valve Command Capture (VCC) board. This board sampled the TTL thruster signals at 6 MHz., and accumulated the results over each 80 Hz. period. The PIL software would sample the VCC board once per 80 Hz. cycle, and fold the results into the thruster models. 6 MHz. was a much higher sample rate than required by this simulation, but this high rate was no more difficult to implement than a lower rate.

This board required a short C interface routine, which was handwritten code. This routine did little more than call the low-level I/O routines supplied by the AC-100, and pass the results along to the block diagram. This routine was implemented in the block diagram as a User Code Block.

## Attitude Dynamics Models

The MSTI 2 spacecraft was modeled as four interacting rigid bodies: The main spacecraft structure and three reaction wheels. The main attitude dynamics block diagram is shown in Figure 5. The equations of motion of these bodies were developed using Kane's method, with the AutoLev software package. Kane's method allowed the models to include all forces of interest on the bodies, including non-conservative frictional forces and arbitrary actuator forces.

The spacecraft structure was modeled as a rigid body with misalignment of the principal axes of inertia. The wheels were modeled as axisymmetric bodies with their axis of symmetry aligned with their spin axis.

The moments and products of inertia of each body were set as parameters in the block diagram.

Using AutoLev, the equations describing the interaction of the bodies was described vectorially, and then AutoLev automatically generated a complete stand-alone implementation of these equations. One of the routines which AutoLev generated computes the algebraic relation between (1) the current state of the system and the actuator forces and (2) the angular accelerations of the bodies.

**Rotational Dynamics Block Diagram**

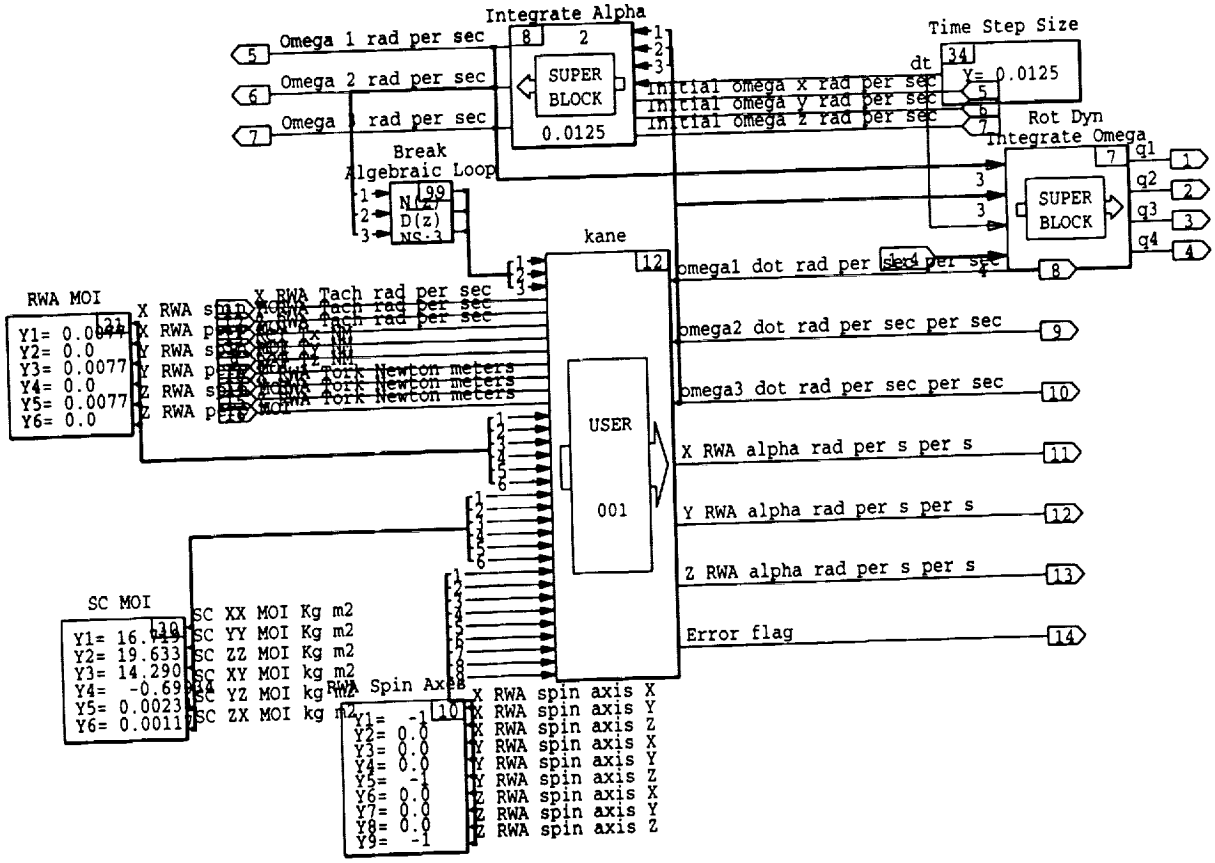


**Figure 5**

This routine was extracted from the AutoLev-generated program, and incorporated into the block diagrams as a User Code Block. The inputs to this routine were (1) the total torque due to the thrusters, (2) the total torque acting on each of the reaction wheels, (3) The current angular velocity of the spacecraft frame, (4) torque acting on each of the reaction wheels, (5) the mass properties of the spacecraft structure, (6) the mass the current angular rate of each wheel, (5) the mass properties of the spacecraft structure, (6) the mass properties of the individual wheels, and (7) the orientation of each wheel in the spacecraft. The outputs from this User Code Block were the angular accelerations of the spacecraft structure and the angular accelerations of the individual wheels.

The angular accelerations were fed into discrete-time integrators to compute the angular rate of the spacecraft and the speed of the wheels. The spacecraft rates were fed into a discrete-time quaternion propagator to compute the spacecraft attitude.

## Orbit Dynamics Models and Sun Ephemeris

These are two high fidelity models which were developed several years ago for other programs. They were developed as handwritten C source code. Sometime after their original development, they were modified to be User Code Blocks for use in the SystemBuild environment. Both of these models were used without

further modification in the MSTI 2 PIL.

The Orbit dynamics model includes a fifth order gravity model, and incorporates accelerations due to thrusting. Its propagator is a fixed-step fourth-order Runga-Kutta integrator. The input to the sun ephemeris is the time, expressed as year, month, day, hour, minute, second, and millisecond, and the output is the sun location in ECI, accurate to a few arcseconds.

User Interface

While the PIL is executing, an interface was presented to the user which allowed the operator to interact with the real-time simulation. This interface was built under Interactive Animation (IA). Using the Interactive Animation editor, the screens were built up graphically and connected to the various inputs and outputs of the block diagram.

**Interactive Animation User Interface**

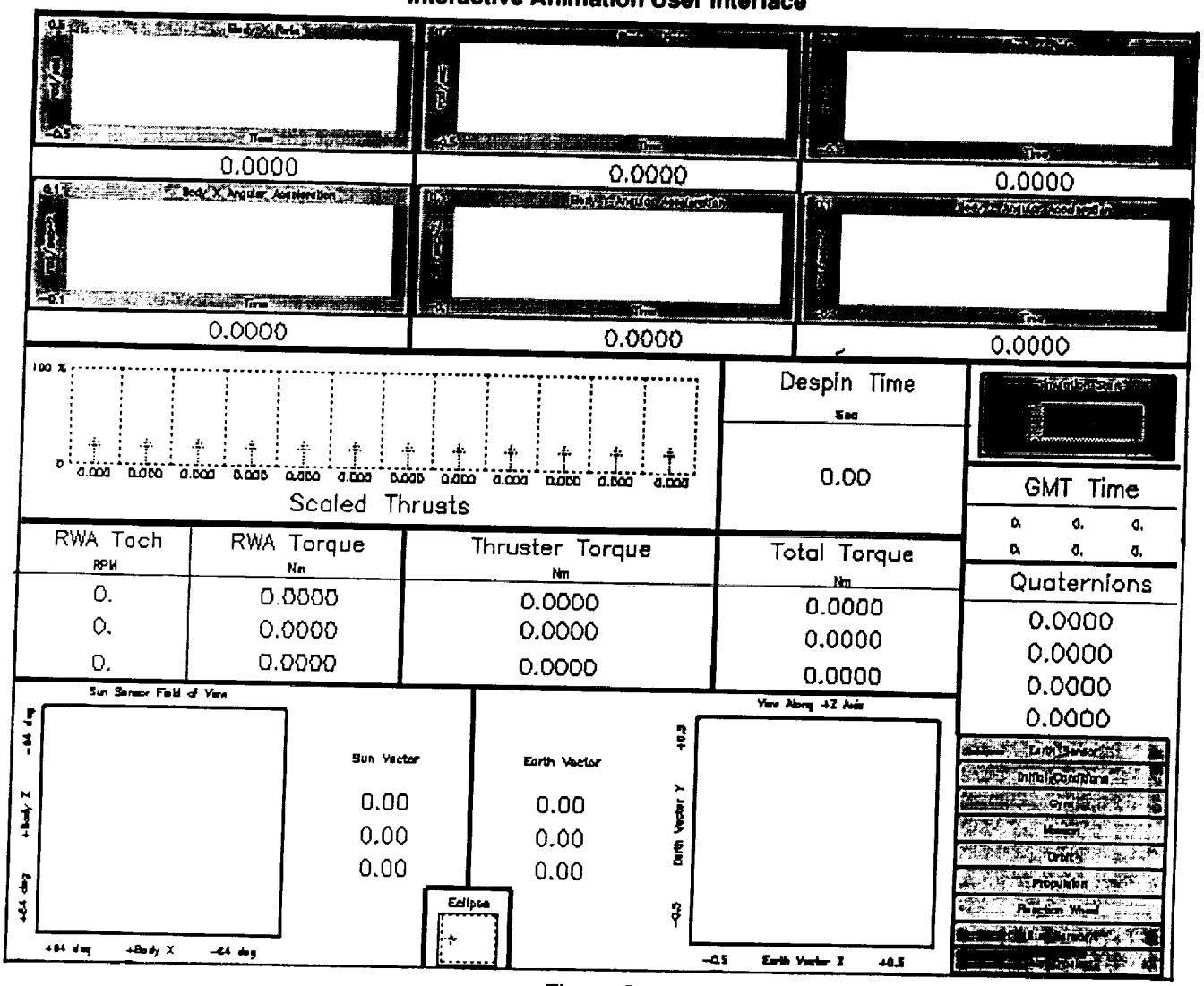


**Figure 6**

The main executive PIL screen is shown in Figure 6. From this screen, the user could invoke any one of many different screens at any time. With these screens, the user could monitor various internal variables in the PIL simulator real-time, or the user could interact with the simulation by varying parameters and adjust-

ing various settings, again in real-time. This capability was used primarily to simulate faults in the spacecraft, in order to test the response of the on-board software to off-nominal conditions.

A different screen dedicated to each subsystem was included in the Interactive Animation interface. With each screen dedicated to an actuator, the user could override the commands coming from the flight processor and inject his or her own commands. The user could command individual thruster firings, and could command reaction wheel torques directly. Similarly, with each screen dedicated to a sensor, the user could override the sensor data computed by the PIL simulation and inject his or her own values. The user could place the sun in any orientation relative to the spacecraft coordinates, could place the earth in any orientation relative to the spacecraft coordinates, or could inject any body rates into the gyro models.

Outside of the flight processor, the variables internal to the on-board software are not available to the spacecraft. In order to keep the PIL a true environment emulator, it also had no access to the variables internal to the processor. Therefore, the Interactive Animation screens could not present all of the information of interest in the spacecraft. Most significantly, the PIL provided no direct method to determine what attitude control mode the processor was currently using.

## Custom Interface Boards

As described above, several custom electronics boards were developed for this project. These boards were prototyped and debugged by the primary engineer, and the artwork and fabrication was subcontracted to an electronics design house.

The boards included the GPC board, and the VCC board. The MSTI 2 PIL was implemented on an older model of the AC-100 which did not have an efficient serial interface, so it was necessary to build a custom serial interface, the Dual Serial Transmitter (DST). In addition, a simple executive board, the ASBX, was build to control the interactions of the other boards with the AC-100.

Because the graphical programming environment of SystemBuild provides such rapid software development, most of the time spent developing the MSTI 2 PIL went toward hardware development.

## Conclusion

By taking maximum advantage of the AC-100 development environment, one engineer spent three months, with one month of help from a second engineer, to develop a high fidelity spacecraft simulator. This included all initial design, all model development, all software development, the design and development of four custom electronics boards, integration of the subsystems, and refinement of the system. This was due primarily to the graphical programming environment of SystemBuild, along with the code generation capability of AutoCode. Another prime factor in this success was the ability of the AC-100 system to reuse existing code.

By spending very little time on software development, the engineer was allowed to focus on the more difficult task of hardware development.